SSC - Communication and Networking

Java Socket Programming (III)

Shan He

School for Computational Science
University of Birmingham

Module 06-19321: SSC
Outline of Topics

Real-time protocol (RTP)

XMPP protocol

Java I/O and New IO (NIO)
Real-time protocol (RTP): What is it?

- Defined by “RFC-1889, RTP, a Transport Protocol for Realtime Applications”
- A standardized packet format for delivering audio and video over IP networks, e.g., to multiple destinations
- **Implementations are built on the UDP. Why?**
- **Question**: How to cope with misordered or lost packets in UDP?
- **Answer from RTP**:  
  - Ignoring the loss or re-sending the missing packets,
  - Providing the ability to process out-of-order packages by adding a sequence number to each packet
Applications of RTP

- IP Phone
- Audio conference
- Video conference
- IP television / video on demand
Java Media Framework (JMF)

- “A Java library that enables audio, video and other time-based media to be added to Java applications and applets.”
- Used for capturing, playing, streaming, and transcoding multiple media formats
- Provides support for RTP to enable the transmission and reception of real-time media streams across the network.
- Cannot be used for Android and lack of updates and maintenance (last update in 2008!)
- A few alternatives exist, the most promising one is Freedom for Media in Java (FMJ)
Further reading

Since JMF and RTP are beyond this module, you can learn from:

- IBM JMF Tutorial
- JMF tutorial from Univ. of Colorado
- Some code examples from Old Dominion University
- Documents for developing RTP APPs in Android
XMPP: what is it?

- XMPP: Extensible Messaging and Presence Protocol
- A open standard communications protocol for message-oriented middleware based on XML
- Was a open-source project called Jabber
- Formalize by Internet Engineering Task Force (IETF) as a open standard, no royalties are required
- Cisco acquired Jabber (Cisco Jabber) in 2008 for Enterprise Instant Messaging
- Can be used for instant messaging, VoIP, video, file transfer, gaming
- Used by Google Talk, Facebook’s chat, Skype, Whatsapp, Apple Messages. Click Full list or here
Smack Java example: chat with GTalk and Facebook Chat

- We can use Smack to send/receive chat messages to/from Google Talk and Facebook Chat
- Not in the campus because of the firewall
- A few tutorials/examples to get you started:
  - Smack official document.
  - A facebook chat client using Smack
  - Facebook Chat with Java
- You can do a lot of cool things using Smack: voice chat, video chat, file transfer, games, etc.
- Learn how to programming online games using XMPP
Java I/O: what is it?

- I/O: input/output, refers to the interface between a computer/the world, or program/the rest of the computer.
- Usually built into the operating system.
- Uses stream metaphor: packaged and transmitted data as one byte at a time, through an object called a Stream.
- Reminder: Socket is a kind of I/O.
- Advantages: easy to process the streamed data through I/O, e.g., to create filters.
- Disadvantages: slow
Java NIO: what is it?

- An alternative Java IO API to the standard Java IO API
- Offer features for intensive I/O operations
- Uses a different metaphor: block I/O
- Block I/O: data is packaged and transmitted in blocks.
- Advantages: faster than standard stream-oriented I/O
- You can read more from Oracle's Java page
- Java 7 release NIO version 2 in 2011: provides extended capabilities for file system tasks,
Java NIO: important concepts

- Two central concepts:
  - Channel
  - Buffer

- Channels are analogous to streams in the original I/O package where you can read and write data.

- A Buffer is a container object, which holds some data, that is to be written to or that has just been read from.

- Buffer is the most significant difference between tradition stream-oriented IO and NIO

- In stream-oriented I/O data is directly written to, and read data from, Stream objects, but in NIO, data is read and write via Buffer
Java NIO: Channels details

- All IO in NIO starts with a Channel.
- Three types of network related channels:
  - DatagramChannel: read/write data via UDP
  - SocketChannel: read/write data via TCP.
  - ServerSocketChannel: similar to ServerSocket to listen for incoming TCP connections
- Also file channel: FileChannel
- Similar to Streams
Java NIO: Differences between Channels and Streams

- Channels are bi-directional: you can both read and write data from/to a Channels.
- Streams are one-way: you need to use `InputStream` and `OutputStream` separately.
- Stream is synchronous or blocking, e.g., when a thread invokes a `read()` from a stream, that thread is blocked until there is some data to read; or when a thread invokes a `write()`, that thread is blocked until the data is fully written.
- Channels can be asynchronous, that is, reading and writing data without blocking. For example, a thread can request reading data from a channel, it will only get what is currently available, or nothing at all, if no data is currently available.
- Channels always read/write data to/from a Buffer.
Java NIO: Buffer details

- Essentially a block of memory, or more precisely, an array.
- But more than an array: provides structured access to data and also keeps track of the system’s read/write processes.
- There are different kinds of Buffer:
  - ByteBuffer: most common one, can be used for most of the I/O operations
  - CharBuffer, ShortBuffer, IntBuffer, LongBuffer, FloatBuffer, DoubleBuffer
Java NIO: How to use `Buffer`?

Five steps:

1. **Step 1: Create a buffer**
   ```java
   ByteBuffer buffer = ByteBuffer.allocate(1024);
   ```

2. **Step 2: Read data from a Channel and then write into the Buffer**
   ```java
   inChannel.read(buffer);
   ```

3. **Step 3: Call `buffer.flip()` to switch the buffer from writing mode into reading mode**

4. **Step 4: Read data out of the Buffer**
   ```java
   buffer.get() or buffer.array()
   ```

5. **Step 5: Call `buffer.clear()` or `buffer.compact()` to clear the buffer. `clear()` clears all buffer, `compact()` only clears the data which you have already read.**
Java NIO: Using FileChannel to read a file.

- Three steps:
  - Step 1: Get a FileChannel from a FileInputStream
  - Step 2: Create a buffer
  - Step 3: Read from the channel into the buffer

- Q: How it differs from original I/O?
- A: Original I/O only creates a FileInputStream and read from that

- Some tricks to make it even faster: use Channel to Channel Transfers
Java NIO network programming: SocketChannel

Steps for setting up SocketChannel

- **Step 1: Open a SocketChannel**
  ```java
  SocketChannel socketChannel = SocketChannel.open();
  ```

- **Step 2: Connect to a server**
  ```java
  socketChannel.connect(new InetSocketAddress("google.com", 80));
  ```

- **Step 3: Reading/writing from/to a SocketChannel to Buffer**

- **Step 4: Close the SocketChannel**
Java NIO network programming: ServerSocketChannel

Steps for setting up ServerSocketChannel

▶ Step 1: Open a ServerSocketChannel
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();

▶ Step 2: Listening for Incoming Connections
while(true) {
    SocketChannel socketChannel = serverSocketChannel.accept();
    //communication using socketChannel
}

▶ Step 4: Close the ServerSocketChannel
serverSocketChannel.close();
Further readings

- Oracle’s tutorial
- KryoNet: API for efficient TCP and UDP client/server network communication using NIO
- Five ways to maximize Java NIO and NIO.2
- You can ignore those bits on non-blocking or asynchronous mode and selection. I will come back to this after multi-threading programming.